A Semantic Parser to Convey Negation Meanings from Natural Language

by

X. Zhang

Supervisor: Michael Gruninger

A thesis submitted for MIE498 Half-year Undergraduate Thesis

Faculty of Applied Science and Engineering

University of Toronto

Abstract

The objective of this thesis project is to create a semantic parser that converts natural language queries into First-Order Logic formulae by utilizing a specified ontology. The parser uses ccg2lambda compositional semantics system and the Natural Language Toolkit (NLTK) Python library to identify the logical formula from a parent-children tree that corresponds to a given natural language input sentence. In this report, we present the development process of a negation template for the semantic parser, including the problems encountered and possible solutions. We evaluate the performance of the template using spatial sentence examples and discuss possible improvement methods for templates developed. The methodology used involves the creation of a syntactic parser using feature-based grammar from NLTK and the development of a .fcfg file containing grammar and lexical rules. The findings demonstrate the effectiveness of the developed template and the potential of using the ccg2lambda system for natural language processing tasks.

Acknowledgements

I am deeply grateful for the invaluable assistance and guidance provided by my supervisor, Professor Michael Gruninger, and the Master student, Yixin Sun, at the Semantic Technology Laboratory. Without their unwavering support, this thesis project would not have been possible. In particular, I would like to express my heartfelt appreciation to Professor Michael Gruninger for providing me with such an outstanding opportunity to expand my knowledge and skills.

Lastly, I would like to extend my utmost gratitude to my family and friends for their constant love and encouragement throughout this academic journey.

Table of Contents

1.0 Introduction

In recent years, there has been significant attention given to natural language question-answering (QA) systems. To translate natural language into query language, semantic parsing is an essential component of these systems [1]. This thesis project aims to solve the challenge of transforming natural language queries into First-Order Logic formulae using a given ontology. However, as the literature demonstrates, existing semantic parsers have typically been evaluated using knowledge databases, which means their reported performance is based on general knowledge rather than the specific knowledge required for a given application. To address this limitation, we decided to create our own semantic parser with different templates. The proposed ontology-based parser utilizes the ccg2lambda compositional semantics system and Natural Language Toolkit (NLTK) Python library to find the corresponding logical formula from a parent-children tree, given a natural sentence input. The output of this parser is in First Order Logic (FOL). To check the performance of our templates, we will use spatial sentence examples from the book "Language, Proof and Logic" by Jon Barwise & John Etchemendy.

This report provides a clear explanation of the methodology used to develop a negation template for the semantic parser. It also describes the problems encountered during the template development process and possible solutions. The future work section includes an assessment of the template's effectiveness and performance and possible improvement method for templates developed.

2.0 Literature Review

To fully grasp the content of this thesis, it is essential to have a solid understanding of the ccg2lambda compositional semantic system and the Natural Language Toolkit (NLTK) Python library, as they serve as necessary background knowledge.

Ccg2lambda is a semantic parser that obtains meaning representations from a Combinatory Categorial Grammar (CCG) tree [2]. It is a popular tool in the field of natural language processing due to its ability to handle different ontologies interchangeably. The system is designed to generate logical semantic representations of sentences in a simple and user-friendly manner. It operates by composing semantic formulas bottom-up using a CCG parse tree [2]. This approach allows for the efficient translation of natural language into logical expressions, making it a valuable resource for a variety of NLP applications. The system's components are illustrated in Figure 1 below. Currently, for this thesis project, our focus is on developing various semantic templates, which correspond to the first half of the flow shown in the figure.

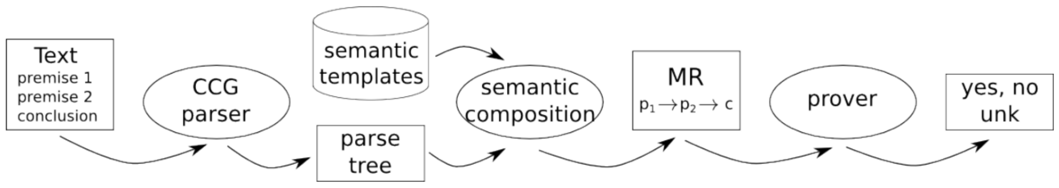
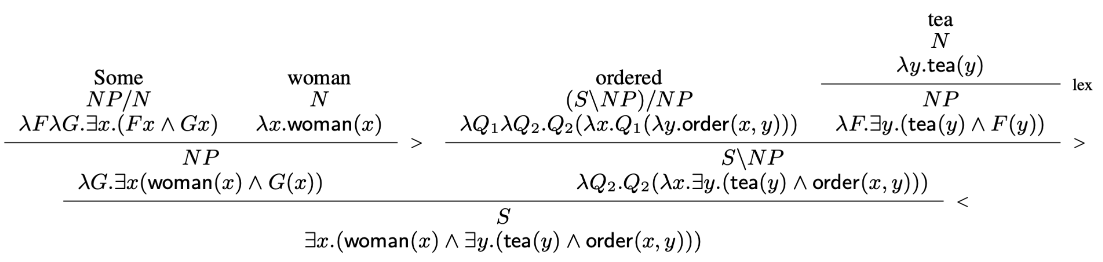


Figure 1: System pipeline for Ccg2lambda compositional semantic system [2]

The system is composed of three stages. In the first stage, the C&C CCG parser, which is a linguistically motivated parser known for its high efficiency, is used to parse sentences into CCG trees. The parser employs a supertagger to assign CCG lexical categories to words in a sentence and can request additional categories if it cannot find a spanning analysis [3]. In the second stage, the system performs semantic compositions by constructing meaning representations (MRs) compositionally over CCG trees using lambda calculus. The semantic composition is obtained from a declarative YAML file, which can be replaced with any ontology [2]. The template requires two attributes: semantics and category. The attribute semantics should conform to NLTK semantics format and be in lambda terms. Syntactic and semantic mappings generate a CCG tree with each node annotated in the format of <tokens>, <ccg>, and <semantics>. An example of parsing sentences into CCG trees is illustrated in Figure 2. The third stage involves theorem construction and the definition of predicate types, which is not the focus of our project and will not be discussed further for now [2].



Some
NP/N
$\lambda F \lambda G.\exists x.(Fx \wedge Gx)$
woman
N
$\lambda x.\mathsf{woman}(x)$
ordered
$(S\backslash NP)/NP$
$\lambda Q_1 \lambda Q_2.Q_2(\lambda x.Q_1(\lambda y.\mathsf{order}(x,y)))$
tea
N
$\lambda y.\mathsf{tea}(y)$
lex
NP
$\lambda F.\exists y.(\mathsf{tea}(y) \wedge F(y))$
>
NP
$\lambda G.\exists x(\mathsf{woman}(x) \wedge G(x))$
>
$S\backslash NP$
$\lambda Q_2.Q_2(\lambda x.\exists y.(\mathsf{tea}(y) \wedge \mathsf{order}(x,y)))$
<
S
$\exists x.(\mathsf{woman}(x) \wedge \exists y.(\mathsf{tea}(y) \wedge \mathsf{order}(x,y)))$

Figure 2: CCG derivation tree and simplified semantics of the sentence "Some woman ordered tea" [2]

Figure 2 provides a detailed depiction of the derivation tree, which encompasses several components. The nodes of the tree comprise CCG syntactic categories that are listed beneath each word in the sentence. For example, the word "woman" is categorized as N (Noun), while "ordered" is categorized as (S\NP)/NP. This particular category is a type-raising CCG category, where the backward slash "\" and forward slash "/" symbols indicate the direction of the function application. Essentially, this category can be understood as a function that takes an NP (Noun Phrase) as its argument and returns a function that takes an S (Sentence) as its argument, ultimately resulting in an NP (Noun Phrase). In this context, it signifies “ordered” as a transitive verb phrase [4]. Beneath the syntactic representations, each phrase is accompanied by a corresponding logical semantic representation, such as λx.woman(x) for the word “woman”. The system operates by taking syntactic categories as input and producing logical formulas as output. In addition, the figure displays the token base forms in the leaves, and the symbols “<”, “>”, and lex (used to specify the combination rule of that inner CCG node) signify the left and right function application rules and type-shift rules in C&C, respectively. These rules and syntactic categories work together to guide the semantic composition process, which is facilitated by semantic templates that describe the specific semantics of the phrases [2]. The actual output of the ccg2lambda system for the sentence “Some woman ordered tea” is presented in Figure 3 below.

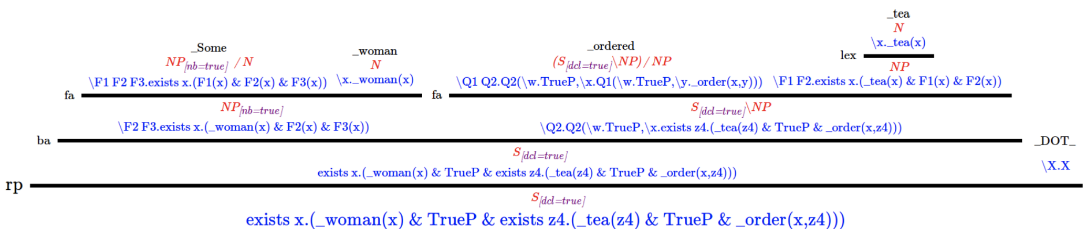
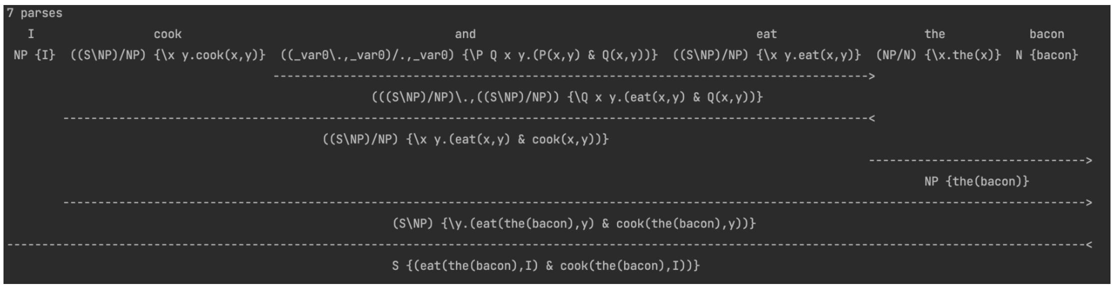

_Some
NP[nb=true] / N
\F1 F2 F3.exists x.(F1(x) & F2(x) & F3(x))
_woman
N
\x._woman(x)
_ordered
(S[dcl=true]\NP) / NP
\Q1 Q2.Q2(\w.TrueP,\x.Q1(\w.TrueP,\y._order(x,y)))
_tea
N
\x._tea(x)
lex
NP
\F1 F2.exists x.(_tea(x) & F1(x) & F2(x))
fa
NP[nb=true]
\F2 F3.exists x.(_woman(x) & F2(x) & F3(x))
fa
S[dcl=true]\NP
\Q2.Q2(\w.TrueP,\x.exists z4.(_tea(z4) & TrueP & _order(x,z4)))
ba
S[dcl=true]
exists x.(_woman(x) & TrueP & exists z4.(_tea(z4) & TrueP & _order(x,z4)))
DOT
\X.X
rp
S[dcl=true]
exists x.(_woman(x) & TrueP & exists z4.(_tea(z4) & TrueP & _order(x,z4)))

Figure 3: Semantic output of ccg2lambda for the sentence “Some woman did not order coffee.” [2]

The category attribute of the semantic template can impose certain requirements on the feature structures of CCG nodes that are generated by the CCG parser. If a CCG node's feature structure and syntactic category correspond with those defined in the semantic template, the template will be applied. To illustrate, suppose the semantic template stipulates an NP[dcl = true] syntactic category. This will match a CCG node with either an NP[dcl = true] or NP[dcl = true, adj = true] category, as illustrated in

Figure 3. Additional attributes can be incorporated into the semantic template to specify more conditions for matching with CCG nodes. The "lex" attribute is utilized in the aforementioned example to specify the combination rule for the inner CCG node [2].

By default, the Natural Language Toolkit (NLTK) is primarily a library for natural language processing (NLP) and not for semantic parsing specifically. However, the NLTK package does provide some tools for working with semantic representations of language, such as the ccgchart parser. The chart parsing tool demonstrates the process of parsing a single sentence with a given grammar and lexicon. The chart parsing tool provides users with the ability to control the parsing algorithm in a flexible manner. At every stage of the algorithm, the user can choose which rule or strategy they want to use, enabling them to try various combinations of strategies, such as top-down and bottom-up, and experiment with different approaches [5]. The parser works by constructing a chart, which is a two-dimensional array of cells that represent the possible constituents of the sentence, and then applying the CCG rules to the chart, combining cells and creating new ones until it reaches the top of the chart, where the entire sentence has been parsed [6]. An example parsing output of the ccgchart parser for the sentence "I cook and eat the bacon" is shown in Figure 4 below.



```
7 parses
   I                 cook                                        and                                       eat                     the            bacon
 NP {I}  ((S\NP)/NP) {\x y.cook(x,y)}  ((_var0\.,_var0)/.,_var0) {\P Q x y.(P(x,y) & Q(x,y))}  ((S\NP)/NP) {\x y.eat(x,y)}  (NP/N) {\x.the(x)}  N {bacon}
                                       ----------------------------------------------------------------------------------->
                                                    (((S\NP)/NP)\.,((S\NP)/NP)) {\Q x y.(eat(x,y) & Q(x,y))}
         -----------------------------------------------------------------------------------------------------------------<
                                             ((S\NP)/NP) {\x y.(eat(x,y) & cook(x,y))}
                                                                                                                            ----------------------------->
                                                                                                                                   NP {the(bacon)}
         ------------------------------------------------------------------------------------------------------------------------------------------------>
                                                       (S\NP) {\y.(eat(the(bacon),y) & cook(the(bacon),y))}
---------------------------------------------------------------------------------------------------------------------------------------------------------<
                                                       S {(eat(the(bacon),I) & cook(the(bacon),I))}
```

Figure 4: Example output of ccgchart parser for the sentence "I cook and eat the bacon"

3.0 Methods and Findings

In the past, we have explored several types of semantic parsers, such as FREYA, PANTO, Pythia, SEMPRE, and ccg2lambda. We used pure first-order logic to ensure that all intended semantics were fully captured. After evaluating the performance of each parser, our team concluded that ccg2lambda is the most suitable foundation for our work, as ccg2lambda allows us to freely swap ontology templates and generate logical formulae in lambda calculus form. However, the C&C parser used in ccg2lambda is deprecated and difficult to set up and operate. Consequently, we decided to create our own template instead of relying on existing ones. This thesis paper focuses mainly on developing the negation template.

To build up a semantic parser, a syntactic parser is crucial, requiring Part of Speech(POS) tags and semantic features for each token. Words can be categorized into different lexical or part-of-speech categories, including but not limited to, Nouns, Verbs, and Articles. These categories are represented by POS tags such as NN (Noun), VB (Verb), and AT (Article). To commence processing the input text effectively, it is necessary to segment it into various linguistic units such as words, punctuation, numbers, or alphanumerics, which are commonly referred to as tokens [7]. Feature-based grammar from NLTK is useful for this process, as it provides an abundance amount of well-developed .fcfg documentation that contains syntactic and lexical rules [8]. To parse a sentence, the program will first split the sentence into separate words and will then loop and search through the grammar file to identify the suitable feature tag, which contains lexical rules for each parsed natural language token. It will then merge their semantic representations based on the predefined grammar rules. The parser will generate two outputs: a grammar parse tree that provides a clearer visualization of the sentence structure and the semantic formula. Figure 5 below shows the general structure of the semantic parsing process.

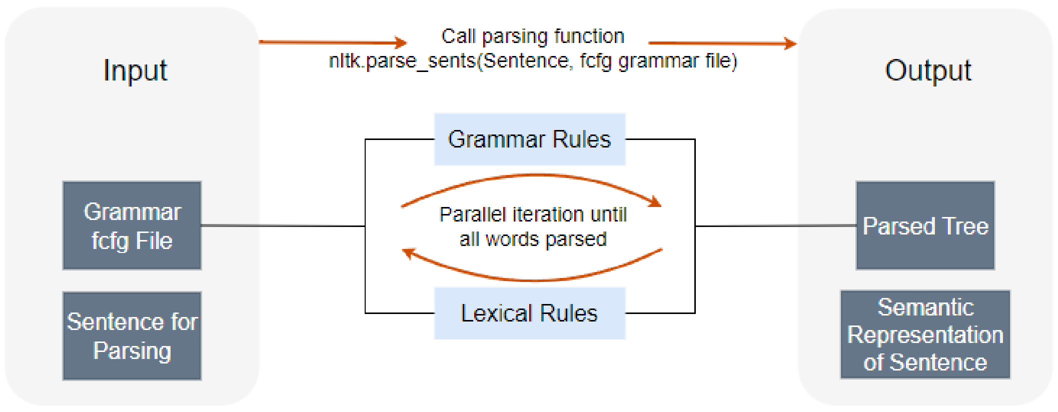

Input
Call parsing function
nltk.parse_sents(Sentence, fcfg grammar file)
Output
Grammar Rules
Grammar fcfg File
Parallel iteration until all words parsed
Parsed Tree
Sentence for Parsing
Lexical Rules
Semantic Representation of Sentence

Figure 5: Semantic parsing process

To test and develop a newly developed template, two files are essential. One is the .fcfg file which records all the grammar and lexical rules defined, the other one is the .py test file for mapping from syntax to semantics.

3.1 Feature-based Context-Free Grammars (fcfg) file

Grammar fcfg file contains two parts: grammar rules and lexical rules.

Grammar rules specify the syntactic categories and features of the elements being combined and including a left-hand side (LHS) and a right-hand side (RHS). The LHS specifies the syntactic category of the element being generated, such as a noun phrase (NP), verb phrase (VP), or sentence (S). The RHS specifies a set of features that define the properties of the element. These features can include tense, aspect, number, gender, case, and other grammatical properties[7][8]. A simple grammar rule template example is demonstrated below:

$$S\,[SEM\ =<?\,subj(?\,vp)\ >]\ ->\ NP[NUM\ =?\,n, SEM\ =?\,subj]\,VP[NUM\ =?\,n, SEM\ =?\,vp] \qquad (1)$$

$$NP[LOC\ =?\,l, NUM\ =?\,n, SEM\ =?\,np]\ ->\ PropN[LOC\ =?\,l, NUM\ =?\,n, SEM\ =?\,np] \qquad (2)$$

$$VP[NUM\ =?\,n, SEM\ =<?\,v(?\,obj)\ >]\ ->\ TV[NUM\ =?\,n, SEM\ =?\,v]\,NP[SEM\ =?\,obj] \qquad (3)$$

$$S\,[SEM\ =<?\,pp(?\,vp)\ >]\ ->\ NP[NUM\ =?\,n, SEM\ =?\,subj]\,VP[NUM\ =?\,n, SEM\ =?\,vp] \qquad (4)$$

Rule (1) can be understood as follows:

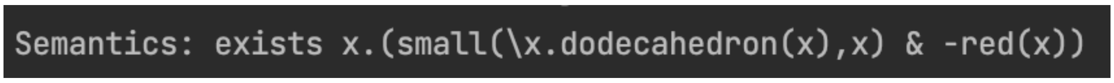
Left-hand side:

- S is the syntactic category, stands for Sentence here.
- The square brackets "[ ]" enclose the feature structure, which is a set of attributes that define the semantics of the sentence.
- SEM stands for "semantic value", which refers to the meaning of the sentence.
- The question mark (?) is a variable that stands for an unknown value, which will be specified later in the rule.
- $SEM\ =<?\,subj(?\,vp)\ >$ is the semantic value of the sentence. This means that the meaning of the sentence is a pair consisting of a subject (*subj*) and a verb phrase (*vp*).

Right-hand side:

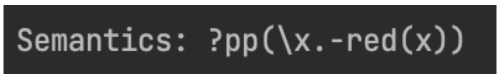
- NP and VP are also syntactic categories, which means they can be generated by other rules in the grammar. For instance, a verb phrase (VP) can also be decomposed into a concatenation of a transitive verb (TV) and another noun phrase (NP), as shown in Rule (3) above.
- NUM is another feature that specifies the number, shown as $NUM\ =\ sg$ (singular) or $NUM\ =\ pl$ (plural) of the noun phrase and verb phrase. It is assigned a value with the equals sign (=).

- The values of the features are specified using the question mark (?) variables, which are then unified with values in other parts of the rule. For example, the value of $?n$ in $NP[NUM\ =?n, SEM\ =?subj]$ will be unified with the value of $?n$ in $VP[NUM\ =?n, SEM\ =?vp]$. It's important to ensure that these values are consistent across the grammar rules. For example, if a sentence contains $NP[NUM\ =\ sg]$, it must also contain $VP[NUM\ =\ sg]$ for the parser to process it correctly. This requirement applies to all grammar rules. For instance, the parser can successfully parse the sentence "a small dodecahedron is not red" but not "a small dodecahedrons is not red", as the NUM value in "a" (singular) is not consistent with "dodecahedrons" (plural) in the sentence."
- $SEM\ =?subj$ and $SEM\ =?vp$ are the semantic values of the noun phrase and verb phrase, respectively. They will be unified with the semantic value of the sentence in the final derivation. By altering a rule, such as changing Rule (1) to Rule (4), the resulting parse tree for a given sentence may differ. For example, the sentence "a small dodecahedron is not red" may produce different parse trees, such as those illustrated in Figure 6 and Figure 7.



```
Semantics: exists x.(small(\x.dodecahedron(x),x) & -red(x))
```

Figure 6: Semantic parsing result with Rule (1)



```
Semantics: ?pp(\x.-red(x))
```

Figure 7: Semantic parsing process with Rule (4)

$$S\ [SEM\ =<?vp(?subj)\ >]\ ->\ NP[NUM\ =?n, SEM\ =?subj]\ VP[NUM\ =?n, SEM\ =?vp] \quad (5)$$

$$S\ [SEM\ =<?subj(?vp)\ >]\ ->\ VP[NUM\ =?n, SEM\ =?vp]\ NP[NUM\ =?n, SEM\ =?subj] \quad (6)$$

Through my experiments, I discovered that the order of the LHS of grammar rules is not important. However, altering the order of features on the RHS can result in a different semantic structure, so one must be careful to avoid errors. For instance, Rule (5) and Rule (1) have the same meaning, while Rule (6) changes the concatenation from $VP + NP$ to $NP + VP$. Rule (5) can successfully parse “Mary chases John.” while Rule 6 can only parse the sentence “Mary John chases.” which has the potential to cause parsing errors.

In summary, the structure of grammar rules is hierarchical. This means that sentences are initially broken down into fundamental phrases, such as NP and VP. The Parts of Speech and cases that form each respective phrase are then specified. Rule (1) can be broken down into Rule (2) and Rule (3), and there are many other grammar rules that provide us with the possibility and flexibility to construct more complex sentence structures by combining different grammar rules.

In order to simplify the understanding and implementation of the grammar rules outlined in the fcfg file, Tables 3.1.1 and 3.1.2 have been included. These tables summarize the meaning of the various syntactic categories and symbols used throughout the template, providing a helpful reference for users.

Table 3.1.1: Syntactic Categories Summary

Symbol	Meaning	Example	Symbol	Meaning	Example
S	Sentence	Mary chases John	$PredN$	Predicative Noun	student
NP	Noun Phrase	A cube	PP	Prepositional Phrase	with telescope
VP	Verb Phrase	chase a dog	Adj	Adjective	small
Det	Determiner	the	Nom	Nominals	small cube

$Conj$	Conjunction	and	$PropN$	Proper Noun	Mary
$PredP$	Predicative Phrase	smells good	N	Noun	dog
IV	Intransitive Verb	bark	P	Preposition	in
TV	Transitive Verb	see	ADV	Adverb	rarely
Aux	Auxiliary	is	Neg	Negation	not

Table 3.1.2: Other Symbols Summary

Symbol	Meaning	Symbol	Meaning
[ ]	enclose the feature structure	SEM	“semantic value”, which refers to the meaning of the sentence
?	a variable that stands for an unknown value	NUM	“number”, is used to indicate whether a noun phrase or verb phrase is singular or plural. The value of the NUM attribute can be "sg" for singular or "pl" for plural
$subj$	a variable to represent the semantic value of the subject of the sentence	LOC	“locative”, is used to indicate the location or place where the action described by the noun or verb phrase takes place
app	function symbol stands for “application”. When we apply the “app” function to $?v$ and $?vp$, we are combining their meanings to get the meaning of the entire verb phrase	COP	“copula” or “linking verb”, is used to indicate whether the verb phrase is a copular or linking verb phrase
prd, l etc.	prd stands for predicate; l indicates the value LOC; n stands for $Noun$; the same logic applies to other syntactic categories	$+$	used to indicate that a feature has a positive or present value. It is often used to specify the presence of a particular feature in a rule
$-$	similar to $+$, it is used to indicate that a feature has a negative or absent value	$\|$	used to specify alternative feature values for a particular feature in a rule

3.2 Lexical Rules

The combination of individual words in a language is governed by lexical rules, which dictate how words are combined with other words or morphemes to form phrases or more complex linguistic structures. These rules modify the argument structures of lexical items, thereby changing their combinatory properties. In this way, it becomes clear that the meaning and syntactic properties of a sentence depend not only on the individual words used but also on how those words are combined. Lexical rules play a crucial role in the generation and analysis of a wide range of linguistic structures in FCFGs, as they specify the properties of individual words and how they can be combined with other words. The existence of lexical rules also allows for the generation of novel sentences that may not have been seen before. With the ability to combine words in a multitude of ways, the semantic parser is able to parse an infinite number of grammatical and meaningful sentences according to the previously established grammar rules [9]. A simple grammar rule template example is demonstrated below:

$$PropN[-\ LOC, NUM\ =\ sg, SEM\ =<\ \backslash P.P(mary)\ >]\ \ ->\ \ 'Mary' \tag{7}$$

$$PropN[-\ LOC, NUM\ =\ sg, SEM\ =<\ \backslash P.P(john)\ >]\ \ ->\ \ 'John' \tag{8}$$

$$TV[NUM\ =\ sg, SEM\ =<\ \backslash X\, y.X(\backslash x.chase(y,x))\ >, TNS\ =\ pres]\ \ ->\ \ 'chases' \tag{9}$$

$$TV[NUM\ =\ pl, SEM\ =<\ \backslash X\, y.X(\backslash x.chase(y,x))\ >, TNS\ =\ pres]\ \ ->\ \ 'chase' \tag{10}$$

Similar to the grammar rule, the lexical rules also have an LHS and RHS, Rule (7) can be understood as follows:

Left-hand side:

- $PropN$ indicates this lexical rule is defined for the proper noun category.
- $-\ LOC$ means The absence of the location feature (not marked with LOC) indicates that the member (word on the RHS) does not have a locational meaning
- $NUM\ =\ sg$ is the singular number feature means that the noun refers to a single entity.
- $SEM\ =<\ \backslash P.P(mary)\ >$ is a lambda calculus expression that represents the meaning of the member. In this case, the expression $<\ \backslash P.P(mary)\ >$ represents a function that takes one argument P and returns the result of applying P to the constant value "mary". The "\" before P indicates that the function takes one argument, which is represented by the variable "$P.$". The

function body $P(mary)$, means that the function applies the argument P to the constant value “mary”, which serves as the value presented in the semantic result. If the body of the function were $P(m)$ instead, then “Mary” would be interpreted as “m” in the final parse tree.

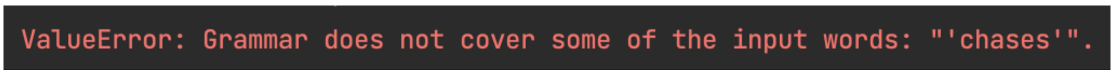
Right-hand side:

- this indicates the word "Mary" is a member of this category $PropN$.

Follow the same logic, Rule (10) can be simply interpreted as follows:

The rule defining the word “chase” is a member of the transitive verb (TV) category with the features of plural number ($NUM = pl$), present tense ($TNS = pres$), and the expression $< \backslash X\, y. X(\backslash x. chase(y, x)) >$ represents a function that takes two arguments (X and y) and returns the result of applying "X" to the lambda function that applies the verb "chase" to the arguments y and x. The "\" before "X" and "y" indicates that the function takes two arguments, which are represented by the variables "X" and "$y.$". The function body, $X(\backslash x. chase(y, x))$, means that the function applies the argument "X" to the lambda function that applies the verb "chase" to the arguments "y" and "x".

In order to prevent value errors, it is crucial to define all words within an input sentence in the corresponding lexical rules. As an example, consider the sentence "Mary chases John." If Rules (7), (8), and (9) are utilized, then the sentence can be parsed successfully. However, if Rules (7), (8), and (10) are used instead, an error will occur, as illustrated in Figure 8 below:



```
ValueError: Grammar does not cover some of the input words: "'chases'".
```

Figure 6: Semantic parsing result for the sentence “Mary chases John” with Rule (7), (8) and (10)

3.3 Tests for mapping from syntax to semantics

Using the code snippet below, we can test the correctness of the grammar rule and lexical rules defined in the fcfg file.

Code Snippet 1: test from syntax to semantics

```
import nltk
#nltk.download('all')
from nltk.ccg import chart, lexicon
from nltk.ccg.chart import printCCGDerivation
from nltk.sem.logic import *
gramfile = 'nltk_data/grammars/sample_grammars/spatial-sem.fcfg'
inputs = ['Mary chases John']
def parsing(new_sent, gram):
  parses = nltk.parse_sents(new_sent, gram)
  for sent, trees in zip(new_sent, parses): #parallel iterations
      for tree in trees:
        print("Parse:\n %s" %tree) #print out the parsing process
        print("Semantics: %s" % nltk.root_semrep(tree)) #print out the parse result
parsing(inputs,gramfile) # call the parsing function to test the parser
```

To parse the sentence, we need to provide the input along with the gramfile which records the path to our fcfg file. By applying Rules (1), (2), (3), (7), (8), and (9), we can obtain the parsing output as shown below:

```
Parse:
```

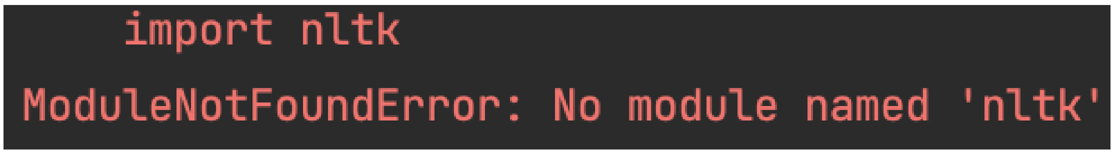
```
(S[SEM=<chase(mary,john)>] # indicates the semantics of the sentence
 (NP[-LOC, NUM='sg', SEM=<\P.P(mary)>] # first level decomposition
  (PropN[-LOC, NUM='sg', SEM=<\P.P(mary)>] Mary)) # found and matches Mary
 (VP[NUM='sg', SEM=<\y.chase(y,john)>] # first level decomposition
  (TV[NUM='sg', SEM=<\X y.X(\x.chase(y,x))>, TNS='pres'] chases)# second level
  (NP[-LOC, NUM='sg', SEM=<\P.P(john)>] #second level
   (PropN[-LOC, NUM='sg', SEM=<\P.P(john)>] John)))) #thrid level, matches John
Semantics: chase(mary,john)
```

The output provides a clear demonstration of the parsing process for the sentence "Mary chases John." The system employs a bottom-up procedure to parse the sentence, following a syntactic tree and exploring all child nodes. At each child node, it searches for the phrase map that matches the current word until it finds the structure that fits the word. Once all words in the sentence have found their corresponding match, the tree is combined, and the final parse result is displayed. This process ensures that the sentence is parsed accurately and efficiently, providing a comprehensive analysis of its underlying structure.

3.4 Problem encountered when creating the negation template

There were many problems encountered during the template development process, below section lists the main problem encountered and possible solutions for them.

1. The nltk module is not working or can not be found as shown in Figure 7.

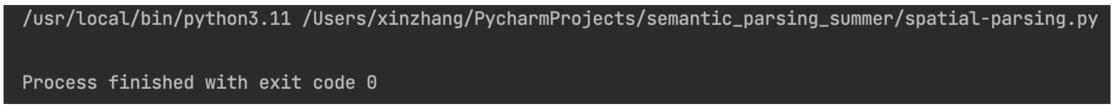


```
     import nltk
ModuleNotFoundError: No module named 'nltk'
```

Figure 7: Error message when importing the nltk module

- One of the main reasons why this error occurs is because the wrong version of the Python interpreter is being used. As you can see in Figure 7, the error is generated when Python 3.7 is used. However, if we switch to Python 3.11, we can resolve this issue.
- Another solution is to manually install the nltk package into the integrated development environment (IDE) that we're using. To ensure that the system is running smoothly, we can simply run the code "nltk.download(all)" to download all the necessary packages.

2. The system did not yield any parsing results, nor did it generate any error messages, as shown in Figure 8 below.



```
/usr/local/bin/python3.11 /Users/xinzhang/PycharmProjects/semantic_parsing_summer/spatial-parsing.py

Process finished with exit code 0
```

Figure 8: No parsing result obtained despite error-free execution

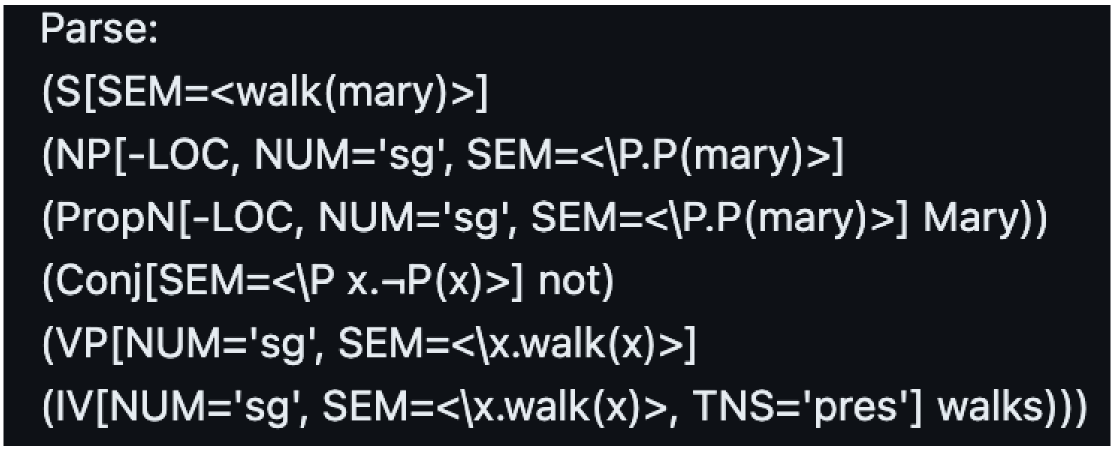
This could happen under serval circumstances:

a) The grammar rule defined is not able to match all words in the input sentence.
    - a potential solution is to manually analyze the sentence structure and cross-check with the grammar hierarchy tree to ensure that the structure can be combined using the existing grammar rules.

b) The input sentence contains grammar errors. For example, the system may fail to parse sentences like "a cube are not a dodecahedron," even though the sentence structure can be combined using the grammar rule defined in the fcfg file.
    - Checking the sentence structure and ensuring the consistency

of features(Tense, Num etc.) will fix the issue.

We also encountered a specific issue related to defining lexical rules with incorrect parts of speech. This problem manifested itself in one of two ways: either the system produced no output (as shown in Figure 8), or it generated incorrect output (as shown in Figure 9).

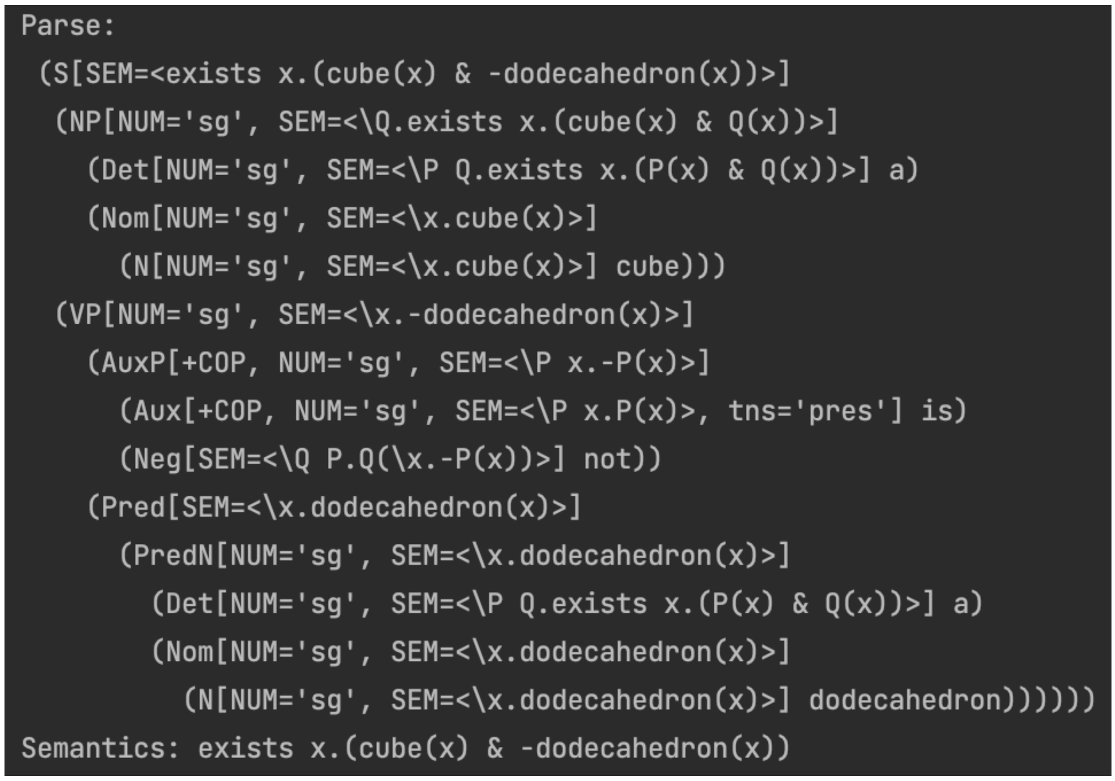


```
Parse:
(S[SEM=<walk(mary)>]
(NP[-LOC, NUM='sg', SEM=<\P.P(mary)>]
(PropN[-LOC, NUM='sg', SEM=<\P.P(mary)>] Mary))
(Conj[SEM=<\P x.¬P(x)>] not)
(VP[NUM='sg', SEM=<\x.walk(x)>]
(IV[NUM='sg', SEM=<\x.walk(x)>, TNS='pres'] walks)))
```

Figure 9: Incorrect parsing output for the sentence “Mary not walk”

In one instance, we defined the part of speech for negation as Conjunction ($Conj[SEM =< \backslash P. \backslash x. (\neg P(x)) >] -> 'not'$) and attempted to parse the sentence “Mary not walk”. The output we received (as shown in Figure 9) indicated that although the system had recognized the presence of “not” in the sentence and matched it with the appropriate lexical rules, it had failed to apply the negation relationship to the verb “walk”. Similar issues arose when we parsed other sentences containing negation. Although the negation sign might appear, it was not always in the correct position. For example, when we parsed the sentence “Mary does not chase John” with the lexical rules for “not” defined as Conjunction, the system produced the result “chase(Mary, -John)”, which is incorrect.

Regrettably, I couldn't determine the reason behind the negation displaying in the incorrect location in the parsing outcome. Nevertheless, I was able to resolve the issue using an alternative approach. I drew inspiration from the ccg2lambda template that was developed on GitHub and attempted to devise a new category known as "Neg" [10]. By emulating existing lexical rules and reassessing the structure of the sentence, the novel template proved to be successful, and it produced an accurate parsing result. The outcome obtained using the new negation template is illustrated in Figure 10.



```
Parse:
 (S[SEM=<exists x.(cube(x) & -dodecahedron(x))>]
  (NP[NUM='sg', SEM=<\Q.exists x.(cube(x) & Q(x))>]
    (Det[NUM='sg', SEM=<\P Q.exists x.(P(x) & Q(x))>] a)
    (Nom[NUM='sg', SEM=<\x.cube(x)>]
      (N[NUM='sg', SEM=<\x.cube(x)>] cube)))
  (VP[NUM='sg', SEM=<\x.-dodecahedron(x)>]
    (AuxP[+COP, NUM='sg', SEM=<\P x.-P(x)>]
      (Aux[+COP, NUM='sg', SEM=<\P x.P(x)>, tns='pres'] is)
      (Neg[SEM=<\Q P.Q(\x.-P(x))>] not))
    (Pred[SEM=<\x.dodecahedron(x)>]
      (PredN[NUM='sg', SEM=<\x.dodecahedron(x)>]
        (Det[NUM='sg', SEM=<\P Q.exists x.(P(x) & Q(x))>] a)
        (Nom[NUM='sg', SEM=<\x.dodecahedron(x)>]
          (N[NUM='sg', SEM=<\x.dodecahedron(x)>] dodecahedron))))))
Semantics: exists x.(cube(x) & -dodecahedron(x))
```

Figure 10: Parsing result for the sentence “A cube is not a dodecahedron”

In a manner similar to the explanation provided in section 3.3, we can observe that the word “not” is being recognized as part of the Auxiliary Phrase along with “is”, while the noun “dodecahedron” is identified as a Predicate Noun. This also demonstrates that a single word may be parsed differently as part of various phases (such as PredN or Nominal Noun for dodecahedron). Therefore, it is necessary to define a diverse set of grammar rules to facilitate the incorporation of various sentence structure combinations. Additionally, it is worth noting that including identical grammar and lexical rules in the fcfg file does not lead to parsing errors.

4.0 Limitations and Future Work

It is evident that the parser design comes with an inherent constraint as the semantic representation is constructed through a comprehensive compilation of grammar and lexical rules, which generates a hierarchical tree. Careful formulation of the grammar rules is imperative to ensure that all children nodes are appropriately linked to their respective parent nodes, thus facilitating upward movement in the tree. The tool's unique feature is the assignment of semantic tags to every node in the tree. However, handling universal quantifiers, conjunctions, propositions, and multivariable situations is an area that requires further investigation in relation to semantic representation.

One area for future work on this project is to continue developing new templates and refining the current negation template. While the current template can handle most cases of negation, such as “not” and “no”, it struggles with certain sentences that contain ambiguous words. For example, when parsing the sentence “Cube A is not red”, the output is $"cube(\backslash x. - large(x))"$, which fails to indicate that the cube is specifically referred to as “A”.

In addition to refining the negation template, integrating it with other templates could also help parse more complex sentences. This could involve incorporating additional templates for specific types of phrases or clauses, or further refining existing templates to handle a wider range of sentence structures.

5.0 Conclusion

This thesis project aims to create a semantic parser that transforms natural language queries into First-Order Logic (FOL) formulae using a given ontology. The proposed ontology-based parser utilizes the ccg2lambda compositional semantics system and Natural Language Toolkit (NLTK) Python library to find the corresponding logical formula from a parent-children tree given a natural sentence input. We concluded that ccg2lambda is the most suitable foundation for their work, as it allows for the efficient translation of natural language into logical expressions. However, the C&C parser used in ccg2lambda is deprecated and difficult to set up and operate, so we decided to create our own template instead. The report focuses on the methodology used to develop a negation template for the semantic parser and also describes the importance of feature-based grammar in processing input text effectively. A detailed explanation of grammar rules and lexical rules is also included in the methods and findings section. Overall, the report provides a clear explanation of the methodology used to develop a negation template for the semantic parser, including the problems encountered during the template development process and possible solutions. The future work section includes an assessment of the template's performance and possible improvement methods for templates developed.

Please note that the spatial-sem.fcfg file provides the template, and the code for testing the template is contained in the spatial.py file.

6.0 References

[1] P. Liang, “Learning executable semantic parsers for natural language understanding,” arXiv.org, 22-Mar-2016. [Online]. Available: https://arxiv.org/abs/1603.06677. [Accessed: 06-Apr-2023].

[2] P. Martínez-Gómez, K. Mineshima, Y. Miyao, and D. Bekki, “CCG2LAMBDA: A compositional semantics system,” ACL Anthology. [Online]. Available: https://aclanthology.org/P16-4015/. [Accessed: 06-Apr-2023].

[3] B. Djordjevic, J. R. Curran, and S. Clark, “Improving the Efficiency of a Wide-Coverage CCG Parser.” [Online]. Available: http://web.archive.org/web/20170810182437/http://www.cl.cam.ac.uk/~sc609/pubs/iwpt07efficiency.pdf. [Accessed: 06-Apr-2023].

[4] “A very short introduction to CCG - school of informatics, university of ...” [Online]. Available: https://www.inf.ed.ac.uk/teaching/courses/nlg/readings/ccgintro.pdf. [Accessed: 06-Apr-2023].

[5] E. Loper and S. Bird, “NLTK: The natural language toolkit,” arXiv.org, 17-May-2002. [Online]. Available: https://arxiv.org/abs/cs/0205028. [Accessed: 06-Apr-2023].

[6] *NLTK*. [Online]. Available: https://www.nltk.org/howto/ccg.html. [Accessed: 06-Apr-2023].

[7] *8. analyzing sentence structure*. [Online]. Available: https://www.nltk.org/book/ch08.html. [Accessed: 06-Apr-2023].

[8] *9. building feature based grammars*. [Online]. Available: https://www.nltk.org/book/ch09.html. [Accessed: 06-Apr-2023].

[9] “Chapter 1 Introduction - York University.” [Online]. Available: https://wiki.eecs.yorku.ca/course_archive/2016-17/W/3611/_media/eecs3611_lecture1.pdf. [Accessed: 06-Apr-2023].

[10] Mynlp, “Ccg2lambda/SEMANTIC_TEMPLATES_EN_EVENT.YAML at master · mynlp/CCG2LAMBDA,” GitHub. [Online]. Available: https://github.com/mynlp/ccg2lambda/blob/master/en/semantic_templates_en_event.yaml. [Accessed: 06-Apr-2023].